2014 47th Annual IEEE/ACM International Symposium on Microarchitecture

NoC Architectures for Silicon Interposer Systems

Why pay for more wires when you can get them (from your interposer) for free?

Natalie Enright Jerger, Ajaykumar Kannan, Zimo Li
Edward S. Rogers Department of Electrical and Computer Engineering
University of Toronto
{*enright, kannanaj, lizimo*}*@ece.utoronto.ca*

Gabriel H. Loh
AMD Research
Advanced Micro Devices, Inc.
gabriel.loh@amd.com

***Abstract*—Silicon interposer technology ("2.5D" stacking) enables the integration of multiple memory stacks with a processor chip, thereby greatly increasing in-package memory capacity while largely avoiding the thermal challenges of 3D stacking DRAM on the processor. Systems employing interposers for memory integration use the interposer to provide point-to-point interconnects between chips. However, these interconnects only utilize a fraction of the interposer's overall routing capacity, and in this work we explore how to take advantage of this otherwise unused resource.**

We describe a general approach for extending the architecture of a network-on-chip (NoC) to better exploit the additional routing resources of the silicon interposer. We propose an asymmetric organization that distributes the NoC across both a multi-core chip and the interposer, where each sub-network is different from the other in terms of the traffic types, topologies, the use or non-use of concentration, direct vs. indirect network organizations, and other network attributes. Through experimental evaluation, we show that exploiting the otherwise unutilized routing resources of the interposer can lead to significantly better performance.

I. Introduction

Die-stacking technology enables the combination of multiple distinct silicon chips within a single package. In the past several years, the industrial adoption of die-stacking technologies has been accelerating [9]. Vertical or 3D stacking takes multiple silicon die and places one on top of the other with through-silicon vias (TSVs) providing inter-layer connectivity. Another die-stacking approach takes multiple silicon die, and "stacks" them side-by-side on a silicon interposer carrier, for example as shown in Fig. 1. Known as "2.5D stacking" [15], this technology is already supported by design tools [22], planned for future GPU designs [18], and is already in some commercially-available products [45], [46]. Section II provides more details and a comparison of the approaches.

A likely and promising application of die-stacking technology is the integration of memory (DRAM) with a multi-core processor [10], [25], [37]–[39]. The increasing core counts of multi-core (and many-core) processors demand more memory bandwidth to keep all of the cores fed. Die stacking can address the bandwidth problem while reducing the energy-per-bit cost of accessing memory. We target systems similar to that shown in Fig. 1, where a multi-core chip is 2.5D-stacked on a silicon interposer alongside multiple vertically-stacked DRAMs [16]. Additional memory outside of the package may also be used (but is not shown in the figure).

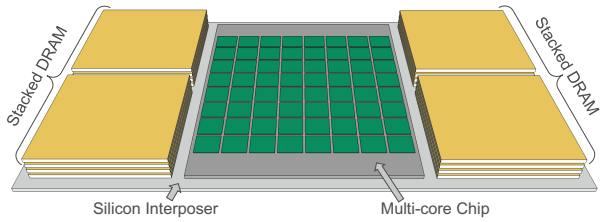

Fig. 1. Baseline system organization consisting of a multi-core chip horizontally stacked with four 3D DRAM stacks on a silicon interposer.

The performance of a multi-core processor is limited not only by the memory bandwidth, but also by the bandwidth and latency of its network-on-chip (NoC). The inclusion of in-package DRAM must be accompanied by a corresponding increase in the processor's NoC capabilities, but increasing the network size, link widths, and clock speed all come with significant power, area, and/or cost for additional metal layers. In this work, we make the observation that the underlying silicon interposer presents significant *under-utilized routing resources* that can be exploited. We propose a *general* hybrid NoC approach matched to the attributes of the silicon interposer and individual chips, and we present a specific example implementation with the following attributes:

- A multi-network topology that blends direct and indirect network approaches, where any-to-any cache-coherence traffic is supported on a subset of the NoC implementing a direct-network topology, while the any-to-few core-to-memory traffic is routed across a subset of the NoC with an indirect topology.
- The physical implementation of the network spans both the multi-core processor die as well as the silicon interposer, with shorter core-to-core links routed across the multi-core processor die and the longer-distance indirect network links routed across the interposer.
- The functional partitioning of NoC traffic is not strict; depending on application needs and actual NoC usage, we load balance traffic to, for example, exploit under-utilized links in the interposer to route cache coherence messages. In some cases, packets can effectively use the longer indirect links on the interposer layer as "express

1072-4451/14 $31.00 © 2014 IEEE
DOI 10.1109/MICRO.2014.61

channels" [13] to reach their destinations in fewer hops.
- Selective concentration is employed to limit the area overheads of vertical connections (i.e., micro-bumps) between the multi-core processor and interposer layers.
- The general approach is applicable to current passive interposers as well as future active interposers.

II. Silicon Interposer Systems

A. 2.5D Stacking Overview

Horizontal or 2.5D stacking [15] enables the integration of multiple chips similar to vertical or 3D stacking [50]. Using Fig. 1 as a reference, a 2.5D system consists of a base silicon interposer with multiple other chips stacked on top. The interposer consists of a regular (but larger) silicon chip, with conventional metal layers facing upward. Current interposer implementations are "passive" and do not provide any transistors on the interposer silicon layer: only metal routing between chips and TSVs for signals entering/leaving the chip [45]. Future generations of "active" interposers could potentially integrate some devices (possibly in an older technology); the approach espoused in this paper is compatible with either type of interposer.

With 2.5D stacking, chips are typically mounted face down on the interposer with an array of micro-bumps (μbumps). Current μbump pitches are 40-50μm, and 20μm-pitch technology is under development [19]. The μbumps provide electrical connectivity from the stacked chips to the metal routing layers of the interposer. Die-thinning is used on the interposer for TSVs to route I/O, power, and ground to the C4 bumps.

The interposer's metal layers are manufactured with the same back-end-of-line process used for metal interconnects on regular "2D" standalone chips. As such, the intrinsic metal density and physical characteristics (resistance, capacitance) are the same as other on-chip wires. Chips stacked horizontally on an interposer can communicate with each other with point-to-point electrical connections from a source chip's top-level metal, through a μbump, across a metal layer on the interposer, back through another μbump, and finally to the destination chip's top-level metal. Apart from the extra impedance of the two μbumps, the path from one chip to the other looks largely like a conventional on-chip route of similar length. As such, unlike conventional off-chip I/O, chip-to-chip communication across an interposer does not require large I/O pads, self-training clocks, advanced signaling schemes, etc.

B. 2.5D vs. 3D Stacking

A limitation of vertical (3D) stacking is that the size of the processor chip limits how much DRAM can be integrated into the package. With 2.5D stacking, the capacity of the integrated DRAM is limited by the size of the interposer rather than the processor. For example, Fig. 1 shows a 2.5D-integrated system with four DRAM stacks on the interposer. Using the chip dimensions assumed in this work (see Fig. 2), the same processor chip with 3D stacking could only support two DRAM stacks (i.e., half of the integrated DRAM capacity). Furthermore, directly stacking DRAM on the CPU chip could increase the engineering costs of in-package thermal management [12], [20], [44].

3D stacking potentially provides more bandwidth between chips. In particular, the bandwidth between two 3D-stacked chips is a function of the chips' common surface area, whereas the bandwidth between 2.5D-stacked chips is bounded by their perimeters. However, 3D stacking incurs additional area overhead for TSVs, often requiring large "keep-out" regions [2], [42], where a 2.5D-stacked chip is flipped face down so that the top-layer metal directly interfaces with the μbumps. However, compared to the conventional approach of going off package for all memory accesses, both stacking options provide a substantial increase in bandwidth at lower energy [9].

C. Opportunities on the Interposer

The interposer simply provides a mechanical and electrical substrate for the integration of multiple disparate chips. Current 2.5D stacking primarily uses the interposer for edge-to-edge communication between adjacent chips (e.g., processor to stacked DRAM). Apart from this limited routing, the vast majority of the interposer's area and routing resources are unutilized. Given the assumption that high-performance systems will use interposers to integrate memory and processors, we consider what else can we do with the interposer? In this work, we propose a general approach to interposer-based NoC architectures that spans both the multi-core processor and interposer layers, exploiting the otherwise *wasted* routing resources of the interposer.

III. NoCs for Interposer-Based Multi-cores

In this section, we first motivate why the NoC should be extended to span across both the CPU die and the interposer. We then address interposer-related implementation challenges. This leads to our overall approach for interposer-based NoCs.

A. Baseline System and NoC

While the approach we propose in this paper is general and broadly applicable to a wide range of interposer-based systems, we will use a working example throughout this paper to make the proposal more concrete. We assume a 2.5D system with a 64-core die with four DRAM stacks as shown in Fig. 2(a). This baseline uses a relatively large interposer, but this still fits within an assumed reticle limit of 24mm$\times$36mm (8.6cm^2). Each of the four memory stacks is assumed to have a size similar to a JEDEC Wide-IO DRAM [23], [27], and we assume four channels per stack. The chip-to-chip and chip-to-interposer edge spacing is assumed to be 0.5mm.

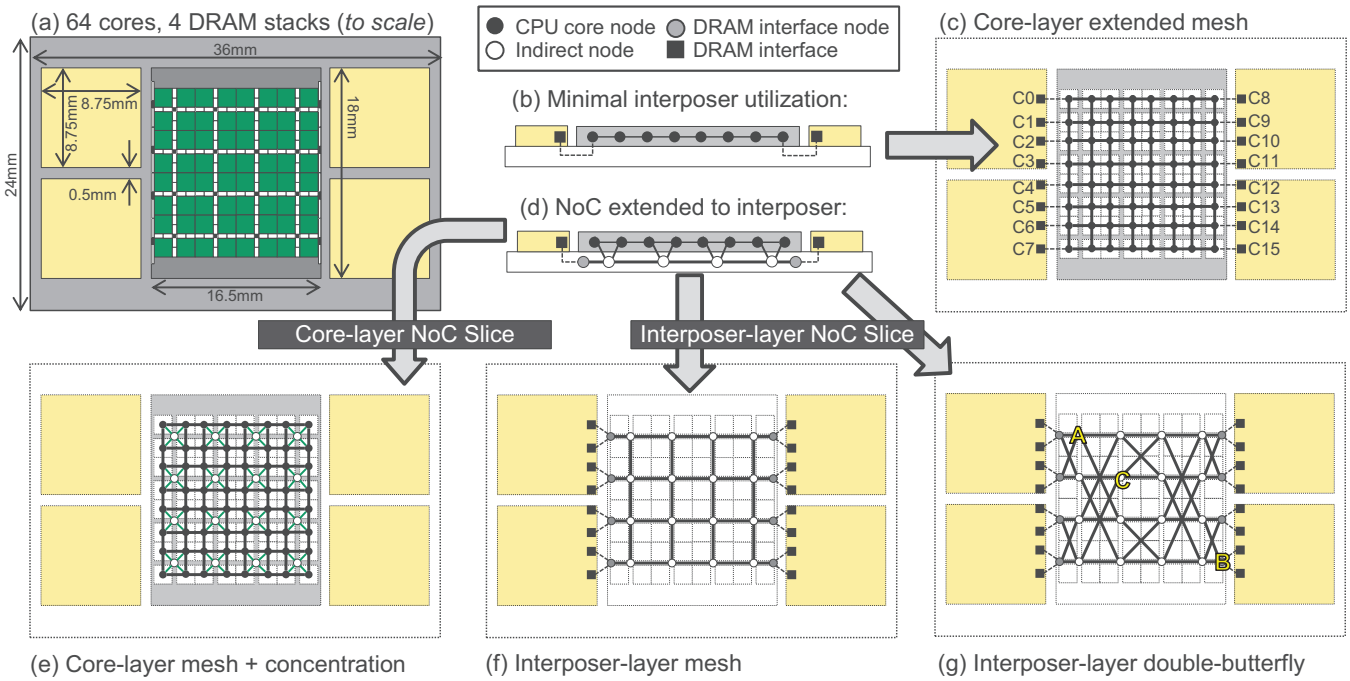

Fig. 2. (a) Top view of the evaluated 2.5D multi-core system with a 64-core CPU chip in the center of the interposer with four DRAM stacks placed on either side of the multi-core die. (b) Side view of a simple interconnect implementation minimizing usage of the interposer. (c) The multi-core NoC slice uses a mesh topology. (d) Side view of a NoC logically partitioned across both the multi-core die and an interposer. (e) Core-layer mesh with concentrated connections to interposer. (f,g) Two different topologies for the interposer NoC slice: concentrated mesh and double butterfly.

B. Physical Implementation Concerns

Current design approaches only utilize the interposer for chip-to-chip routing and vertical connections to the package substrate for power, ground, and I/O [46]. Fig. 2(b) shows a side/cross-sectional view of a system using such an approach; the interposer-layer routing only connects the edges of the multi-core chip with the DRAM stacks. This figure also helps to highlight how little of the interposer's routing resources are utilized with such a minimal design.

In this work, we propose to make use of the abundant and otherwise unused routing resources on the interposer layer to implement a system-level NoC (as opposed to a NoC primarily limited to the multi-core chip). Fig. 2(d) illustrates the basic concept applied to the baseline interposer. The NoC now effectively uses a 3D topology that spans both the multi-core die and the interposer. If an active interposer is used, then the interposer implements both the router logic and wires for the portion of the NoC that resides on the interposer layer. For a passive interposer, the logic to implement the routers remains on the CPU layer, but the wiring goes through the interposer (explained in the following section).

Dealing with a Passive Interposer: To implement the NoC on an active interposer, we simply place both the NoC links (wires) and the routers (transistors) on the interposer layer. Fig. 3(a) shows a small example NoC with the interposer layer's partition of the NoC completely implemented on the interposer. For the near future, however, it is expected that only passive, device-less interposers will be commonly used. Fig. 3(b) shows an implementation where the active components of the router (e.g., buffers, arbiters) are placed on the CPU die, but the wide NoC links (e.g., 128 bits/direction) still utilize the interposer's routing resources. This approach enables the utilization of the interposer's metal layers for NoC routing at the cost of some area on the CPU die to implement the NoC's logic components.[1] Both NoCs in Fig. 3 are topologically and functionally identical, but have different physical organizations to match the capabilities (or lack thereof) of their respective interposers.

We assume a μbump pitch of 45μm [46]. For a 128-bit bi-directional NoC link, we would need 270 signals (128 bits for data and 7 bits of sideband control signals in *each* direction) taking up 0.55mm^2 of area. For a passive interposer, if the interposer layer were used to implement a mesh of the same size as the CPU-layer (i.e., 8×8), each node would need to have four such links (for each N/S/E/W direction). The total area overhead for the μbumps assuming a 64-core chip would be 140mm^2, or nearly half (47%) of our assumed 16.5mm×18mm multi-core processor die.

To reduce the μbump area overheads for a passive-interposer implementation, we use concentration [5]. Every

[1] If the "interposer links" are too long and would otherwise require repeaters, the wires can "resurface" back to the active CPU die to be repeated. This requires some additional area on the CPU die for the repeaters as well as any corresponding μbump area, but this is not significantly different than long conventional wires that also need to be broken up into multiple repeated segments.

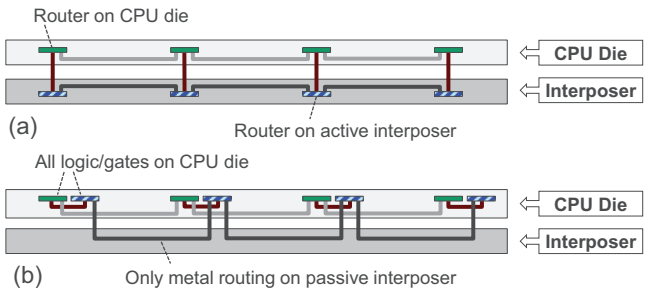

Fig. 3. (a) Implementation of a NoC with routers on both the CPU die and an active interposer, and (b) an implementation where all routing logic is on the CPU die, and a passive interposer only provides the interconnect wiring for the interposer's portion of the NoC.

four nodes in the CPU layer's basic mesh are concentrated into a single node of the interposer-layer NoC. Fig. 2(d) and Fig. 2(e) show different views of this. The side view illustrates the interposer nodes as logically being on the interposer layer (for the passive interposer case, the logic and routing are split between the CPU die and the interposer as described earlier). Usage of a concentrated topology for the interposer layer provides a reduction of the μbump overheads by a factor of four, down to 35mm^2 (or a little less than 12% of the CPU chip area). The area reserved for μbumps is represented by the white boxes in Fig. 2(a). The μbump area overhead is the same for both passive and active interposer cases.

Alternatives: If a passive interposer is only being used to implement the wiring of the second layer of the NoC, a fair question to ask is why not simply route *all* of the NoC links on the CPU die? A chip is typically not implemented with any more metal layers than necessary. As such, a given die will have a ceiling on its bisection bandwidth that is a function of its metal layer count and density. We assume that the baseline core-layer NoC is already interconnect limited (otherwise remove metal layers to reduce cost until it is interconnect limited again). Therefore, our options are to (1) do nothing and accept the bandwidth of the baseline topology of Fig. 2(b), (2) increase the metal layers in the CPU die to support the desired additional bandwidth at a higher cost, or (3) make use of the otherwise available metal layers in the interposer. In effect, to increase NoC bandwidth in an interposer-based system without enduring the cost of additional metal layers in the CPU die, one *must* make use of the interposer. This argument for using the interposer's metal layers also applies to active interposers.

It is important to note that even in our baseline, we assume that an interposer is *already* being used for the purposes of massively increasing in-package integration of memory. In effect, the additional metal resources on the interposer are already "paid for," and we are proposing to exploit these paid-for but underutilized resources to design a better NoC.

C. A Tale of Two NoCs

The distribution of the NoC over two separate layers of silicon, along with the area constraints of the μbumps used to connect these layers, has guided our example interposer-based NoC to employ a conventional mesh-like topology on the CPU die with a concentrated network organization on the interposer layer. At a high level, asymmetry in a NoC could be viewed as undesirable because it makes the overall network less regular, more difficult to route, and potentially introduces unbalanced or unfair levels of service between different parts of the network. Below, we explain that the asymmetry, when properly utilized, can be *advantageous*; we exploit asymmetry between the CPU and interposer layers in a variety of ways.

Differentiating Coherence and Memory Traffic: Differentiating between traffic classes in the network can have numerous benefits. Coherence traffic must be divided across multiple virtual or physical networks [49], [52] in order to avoid protocol-level deadlock. This separation also provides opportunities to tailor the network to particular aspects of the coherence traffic [48]. The core-to-core cache coherence traffic and the core-to-memory traffic exhibit different characteristics [4]. Coherence traffic is characterized by any-to-any communication patterns that over long intervals can resemble uniform random traffic. However, memory traffic produces a many-to-few traffic pattern [1], as memory requests originate from cores and always target memory nodes, never other cores. With memory positioned at the edge of the system, the average hop count for memory traffic on a conventional mesh is substantially larger than the average hop count for cache traffic. Furthermore, cache coherence traffic can often interfere with main memory traffic, and likewise main memory traffic can get in the way of coherence traffic. Given a NoC topology that is distributed across both the multi-core die and the interposer, we propose to functionally partition the NoC so that core-to-core coherence traffic is routed on the multi-core die's portion of the NoC, and the main memory traffic is transported on the interposer's portion of the NoC.

While past work has proposed various ways to partition a NoC based on request types [40], [48], the key difference here is that an interposer-based system can implement a physically separate partition or slice of the NoC without incurring the cost of additional metal layers, while avoiding the contention penalties associated with multiplexing the same physical network across a larger number of functionally-partitioned virtual channels.

Concentrated/Unconcentrated Networks: Due to μbump area constraints, we propose the usage of a concentrated sub-network in the interposer layer. This creates an overall organization where part of the NoC is concentrated (the interposer portion) and part of it is not (the CPU layer), but this asymmetry turns out to be advantageous when coupled with the functional separation of coherence and main-memory

traffic. The concentration results in a smaller diameter for the interposer's network that reduces the average hop count for memory-bound requests to reach their destinations.

Hybrid Direct/Indirect Networks: Each of the individual cores has a link to one of the corresponding routers of the CPU-layer mesh. As such, the portion of the NoC on the CPU layer implements a direct network (i.e., every node of the mesh can source or sink traffic from/to the cores). However, for the interposer's portion of the NoC, only the end points along the left and right edges interface to the die-stacked memory channels, and all other intermediate nodes simply forward packets across the interposer or back up to the CPU layer's mesh. That is, the portion of the NoC on the interposer is an indirect network.

Different Topological Choices: Once we observe that the interposer layer implements an indirect network, it is natural to consider other indirect topologies. Fig. 2(f) shows the straight-forward extension of the original mesh approach to a concentrated, indirect, mesh topology on the interposer layer. We also consider a butterfly-based topology for the interposer layer as shown in Fig. 2(g), which consists of two reflected copies of a classic butterfly network with additional cross links in the middle column (the figure shows diagonal links for ease of illustration, but our evaluations assume standard Manhattan routing of all wires).

Both the concentrated mesh and this double-butterfly have four interposer-layer links per router (and the same number of vertical links back to the CPU die), and so the router costs for both will be very similar. However, using the double-butterfly topology for memory-bound traffic reduces the average number of hops to get to the memory channels. For the example in Fig. 2(g), the longest route from any internal node to one of the exit nodes (left and right columns) is four hops (e.g., A→B). Contrast this to the concentrated mesh in Fig. 2(f) with a worst-case path length of seven hops. As this butterfly has twice the bisection bandwidth compared to the concentrated mesh, we also compare against a non-concentrated mesh with the same bisection bandwidth (but would otherwise be unimplementable in a passive interposer due to the μbump overheads discussed earlier).

From the perspective of a conventional core-to-core NoC design, this double-butterfly would not be desirable because path lengths between nodes internal to the network (i.e., those corresponding to the cores) are sometimes *longer* than they would be compared to the concentrated mesh. For example, the path A→C requires three hops, whereas the equivalent route in the concentrated mesh would only be two. Furthermore, the routing decisions become more complex, as paths need to "double back" (e.g., a path for A→C goes east from A, then south-east, and then *back* west to reach C). However, these sub-optimal paths are not a concern for our system, because the interposer layer's primary purpose is to route memory requests, *not* core-to-core traffic.

The concentrated mesh and butterfly-based topologies are only two possible organizations for the interposer layer's portion of the NoC. Our general approach is not limited to these topologies, but we chose these for our studies because the concentrated mesh follows naturally from the baseline mesh used on the CPU layer, and the butterfly approach provides an effective and illustrative example for how an indirect topology with longer links can be better suited to the memory traffic. Other topologies such as MECS [17], flattened butterflies [28], [29], fat trees [35], Clos or Beneš networks [6], [11], [30], etc., could all be employed depending on the number of cores, the number and layout of the DRAM stacks, etc. The selection of another topology would not fundamentally impact our overall approach nor change any of our conclusions. We provide a comparison with some other topologies in Section VI-D.

D. Routing

Basic Routing: We leverage standard dimension order routing in the mesh and concentrated mesh networks. For the double-butterfly network, we develop a variant of destination-tag routing. In a standard butterfly, the bits of the destination can be used at each stage to select the output port for that stage. Three bits are used to encode the destination memory channel with an additional bit to indicate if the request is destined for a memory channel on the right or left side of the system. There are three cases to consider for routing in the butterfly network; looking at the example in Fig. 2(g), case one is illustrated by routing from node C to the left-hand memory stacks; case 2 is illustrated by routing node C to the right-hand memory stacks and case 3 is illustrated by routing node A to the lower left memory stack. Case 1 uses standard destination tag routing. Each bit that encodes the destination memory channel is used to select the output port at each hop. In case 2, an extra bit of routing information is required for each hop traversed prior to crossing the bisection (one bit in the case of node C). This bit gives the first output port needed to cross the bisection. Once a packet has crossed the bisection, destination tag routing is employed. The routing table supplies the required extra bits based on the destination. In case 3, routes must divert to the previous stage to reach the lower half of the memory channels on the left side. These packets require an extra bit of routing information to route backwards one stage. At this point, destination tag routing is employed.

Our baseline implementation of a NoC spanning both the CPU die and the interposer uses a straightforward traffic partitioning scheme. All core-to-core coherence traffic is routed on the CPU die's portion of the NoC, and all traffic to/from memory goes across the interposer. For requests going to memory, the NoC forwards the packet vertically down to the interposer layer at the first chance possible.

Load-balanced Routing: Some workloads have a larger quantity of core-to-core coherence traffic, while other workloads exhibit higher demands on main memory. If a workload

tends to have a strong bias toward one type of traffic over the other, this may cause underutilization on one of the two layers of the NoC. We also consider routing schemes that dynamically load balance across the two layers. Our load-balancing algorithm will by default attempt to route traffic on its preferred layer (coherence on the CPU layer, memory on the interposer layer). However, if the CPU layer has too much traffic, as measured by tracking the latency of recent messages received by the sending node, and the interposer layer has spare capacity, then coherence requests may be routed to the interposer layer to help alleviate contention in the CPU layer. Tracking observed latency at each node gives a general picture of the state of the network; it does not perfectly reflect congestion on the path of a particular message and may be out of date as network status can change rapidly. However, it is simple to track and provides a reasonable approximation.

Express Routing: The interposer's double-butterfly topology was selected to quickly route memory requests to their destination DRAM stacks. There exists some pairs of cores where the double butterfly offers a *shorter* path via its longer diagonal links. For requests between such pairs of cores, we consider using some of the double butterfly's connections as *express* links [13] to enable longer-distance core-to-core messages to cross the chip in fewer hops. If a core-to-core request can be routed through the interposer with fewer hops than through the core network, it will be sent to the interposer regardless of network load. Certain restrictions are placed on which source-destination pairs can take advantage of express links to prevent deadlock (discussed next).

E. Implementation Concerns

Deadlock Avoidance: Many conventional NoCs leverage turn-model routing to achieve routing-level deadlock freedom. Dimension-order routing (DOR) is the most common example of a turn-model routing algorithm. To avoid routing cycles, turns from the Y to X dimension are prohibited. To provide deadlock freedom in the load-balanced double butterfly, we apply a similar strategy. Routing coherence traffic in the double-butterfly network that would require turns that *double-back* are forbidden. These routes can quickly form network cycles and lead to deadlock. As a result, only a subset of source-destination pairs are eligible for load balanced routing. Multiple VCs are used to avoid protocol-level deadlock from developing between request-response pairs.

Generality to Other Layouts: Our assumed system organization places the DRAM stacks on the left- and right-hand sides of the interposer. However, it is feasible for other system designs to, for example, surround the multi-core die with DRAM stacks on all four sides. Our butterfly-based topology is designed for "east-west" traffic, and it may not be as well suited to a system with a "four-sided" memory layout. However, the key attributes of our overall approach (i.e., functional partitioning of traffic between the CPU die and the interposer, load balancing between layers, choosing an interposer-layer topology that contains longer links to shortcut memory requests to the chip's edges, and reusing long interposer links for express channels) are directly applicable to other interposer-based system configurations and layouts.

IV. Methodology

We use a cycle-level network simulator [24]. All configurations use an 8×8 mesh for the multi-core die. For the interposer's NoC slice, we compare the double-butterfly (DB) topology against a concentrated mesh (CMesh) and a non-concentrated mesh (Mesh). Most comparisons will be focused on the CMesh, as it and the DB have the same router complexity and μbump area overheads. Because the CMesh has only half of the bisection bandwidth compared to the DB, we also make some comparisons to the Mesh, which has the same bisection bandwidth as DB, but requires an impractically large number of μbumps for passive interposer implementations. There are two DRAM stacks on each of the two sides of the interposer. Each DRAM stack provides four independent memory channels, for a system-wide total of 16 channels. The configuration parameters for each topology are listed in Table I. The interposer network dimensions include the end-nodes that interface with the DRAM memory channels (grey circles in Fig. 2).

Our load-balancing design compares recently observed network latencies for the interposer and CPU die networks to determine if some traffic should be offloaded. If the latency in the core die exceeds the latency in the interposer die by a certain threshold, traffic is routed via the interposer die. A lower observed latency is used as a proxy for spare capacity. Empirically, we select a threshold of 10 cycles. This is meant as a proof of concept for inter-layer load balancing; a more complex balancing scheme could be employed. Recent work [14] proposed various metrics to determine when network load should be offloaded to a second network; these techniques could be adapted to interposer-based systems.

To evaluate the baseline and proposed NoC designs, we consider several traffic workloads to cover a wide range of network utilization scenarios. These include many combinations of the traffic patterns described in Table II where core-to-core coherence traffic and requests to memory have different distributions. We consider two hotspot patterns, *UpperLeft* and *Corners*. The channel numbers (e.g., C1) used by the descriptions in Table II refer to the memory channel annotations in Fig. 2(c). Apart from results showing latency versus injection-load curves, we use a batch traffic simulation for multiple request-reply pairs. Each core can support a maximum of four outstanding requests. Traffic is split equally between reads and writes.

We also verified our results with a few PARSEC applications [7] on gem5 [8] with Booksim [24] to model the NoC.

TABLE I. Simulation parameters

Common Parameters (all routers)	
VCs	2, 8 flit buffers each
Pipeline	2 stages
Multi-core Die NoC Parameters	
all configs	8×8 mesh, DOR routing
Interposer NoC Parameters	
Mesh	10×8, DOR routing
Concentrated Mesh	6×4, DOR routing, 4:1 concentration
Double Butterfly	6 stages, 4 routers/stage, Extended Destination Tag Routing, 4:1 concentration

TABLE II. Workloads

Core Traffic (all workloads)	
Uniform random between all core pairs	
Memory Traffic	
Uniform	Uniform distribution of memory references across all 16 memory channels
UpperLeft	12.5% of traffic to each of channels C0, C1 C2, C3; 50% of traffic uniformly distributed across the remaining 3 stacks
Corners	12.5% of traffic to each of channels C0, C7 C8, C15; 50% of traffic uniformly distributed across the remaining 12 channels
Bisection	All Memory references cross bisection
Permutation	Each core sends to only one memory channel (10 random trials)
Percentage of Traffic Memory Reference	0, 25, 50, 75, 100
Read Requests & Write Replies	1 Flit
Data packets	5 Flits
Batch Simulation	1000 requests per core
Max outstanding req.	4 per core

The overall trends are consistent with those observed from our synthetic workloads. Due to the exorbitant run-times required to simulate a 64-core system with a large NoC (88 nodes for CMesh and DB, 136 nodes for Mesh), we focus our experimental analysis on the workloads presented in Table II. To further verify the results using application behavior, we run multi-programmed SynFull [3] traffic patterns. We simulate a 4-way multitprogrammed mix of 16-way multi-threaded applications. Each application is assigned a 4×4 quadrant of cores but its memory accesses are spread across all memory channels. We construct workload combinations based on the intensity of their memory traffic.

Power and area are modeled using DSENT [47]. Results are collected using a 32nm bulk/SOI low-V_t process node with 3.4mm^2 cores, with a worst-case CPU-layer core-to-core link length of 2.2mm (1.8mm CPU width, plus 0.4mm for μbump area). Frequency was swept to determine the maximum operating point for each topology. Long link lengths are faithfully modeled across each topology. μbump power was computed using the activity factors measured from simulation, a system voltage of 0.9V, and a μbump capacitance of 0.7fF [21].

V. Design Comparisons

Before presenting simulation results, this section provides a higher-level comparison of the different interposer NoC topologies. We first provide a qualitative comparison of the topological options based on network/graph characteristics. Table III lists the node-count, router degree (maximum number of ports in a router), network diameter (longest path in the network), average number of hops to memory, total number of bidirectional links (vertical links to the multi-core die are not included as all topologies have 64 such links for the 64 cores), the bisection bandwidth (normalized by the number of bidirectional links), and the physical link lengths for the network configurations assumed throughout this paper. For each attribute, the topology(ies) with the best value is shown in bold in the table.

TABLE III. Network characteristics for the three interposer NoC topologies.

Topology:	Mesh	CMesh	Double Butterfly
Router Nodes	80	**24**	**24**
Router Degree	**5**	8	8
Diameter (in hops)	16	8	**5**
Avg. Memory Distance	7.13	3.75	**2.75**
Total Links	142	**38**	40
Bisection B/W	**8**	4	**8**
Link Length (mm)	**2.2**	4	4, 8, 12

For the most part, the Double Butterfly (DB) is as good, or better, than the best of either Mesh or CMesh. Compared to Mesh, both CMesh and DB have a significantly lower total node count due to concentration at the cost of increasing the number of router ports, but the savings in the total number of routers, as well as the practical implementability concerns related to μbump overheads for passive interposers, are significant. DB has a lower network diameter than the other two alternatives, which in turn is reflected by the average number of hops through the interposer's slice of the NoC to reach a request's destination memory node. Based on total link count, both CMesh and DB are much lower than Mesh, with DB requiring only two more links than CMesh. CMesh has a lower node count than Mesh, but this comes at the expense of half the bisection bandwidth. However, DB enjoys the benefits of concentration while still providing the same bisection bandwidth as Mesh despite having no more router ports than CMesh and having significantly fewer total links than Mesh. Not surprisingly, DB has longer links (the multiple lengths listed correspond to the three different link lengths illustrated in Fig. 2(g)); this ultimately impacts the maximum achievable clock frequency of the router, which is explored in more detail below.

Some of the design trade-offs made for the DB have implications on the physical design of the network and router nodes. For example, the higher port counts of CMesh and DB result in larger routers (more area per router, higher gate counts) that impacts critical timing paths. For each topology, Table IV lists the maximum possible frequency, and the power consumption of the network at the maximum frequency and when all topologies are compared at the same clock speed.

TABLE IV. Comparison of topologies for core-die and interposer's NoCs.

Topology	Maximum Frequency	Power at Max Freq.	Power at 1.75GHz	μbump Power
CPU-layer Mesh	5.05 GHz	6.14W	2.16W	–
Mesh	5.05 GHz	10.37W	4.12W	11.4 mW
CMesh	4.00 GHz	5.95W	2.62W	6.1 mW
Double Butterfly	1.75 GHz	2.96W	2.97W	4.5 mW

DB has a lower maximum clock speed compared to the other topologies due to its longer wires. However, the achievable clock speed of 1.75GHz is still sufficient to keep up with main memory,[2] and clocking the NoC any faster (such as at the maximum possible speeds for Mesh/CMesh) would not likely provide much additional benefit. Table IV also lists the power consumption of the networks at max- and iso-frequency. Naturally, DB consumes less power than the others at maximum frequency simply due to its lower top speed. When all networks are clocked at the same speed, both CMesh and DB achieve lower power levels than Mesh. Mesh power consumption is hurt by a larger number of routers and high hop count. The higher power consumption of the long links in DB is offset by the lower average hop count compared to CMesh.

All systems have the same mesh network on the CPU die; the power consumption of this mesh is also shown in Table IV. Note the CPU-layer mesh has lower power than the interposer mesh due to fewer routers (64 vs. 80) and lower average hop count (any-to-any traffic vs. core-to-memory). Power data is collected using a 0.1 injection rate using the uniform memory distribution workload. The interposer NoC power also includes the power consumption of traversing the μbumps for each hop through the network. This power is negligible compared to the rest of the network power (the μbump capacitance is about three orders of magnitude lower than the interposer wire capacitance). For completeness, Fig. 4(a) shows the breakdown including both the CPU and interposer layers' portions of the NoC power.

Fig. 4(b) shows the area for each network normalized to CMesh. Only the interposer layer's area costs are shown, as all systems have the same CPU layer network. These results use the same workload and system assumptions as used for Table IV. We separate active area from the wiring area of each network. Both CMesh and DB require longer wires than Mesh; however, wiring resources in the interposer are abundant, and so this "cost" does not have a significant impact on the overall system design (i.e., these interposer wiring resources would have been otherwise unutilized). The cost of implementing these networks in the CPU layer could be

[2]The NoC's 128-bit link width matches the stacked DRAM channel's 128-bit data bus width, the number of channels on one side of the interposer (eight) matches the number of uni-directional links along the bisection (or equivalently, the 16 total channels match the bandwidth provided by the eight bi-directional bisection links), and the router clock speed (1.75GHz) is slightly faster than the assumed memory bus speed (1.6GHz). For different memory bandwidth assumptions, other asymmetric and indirect network topologies could be used.

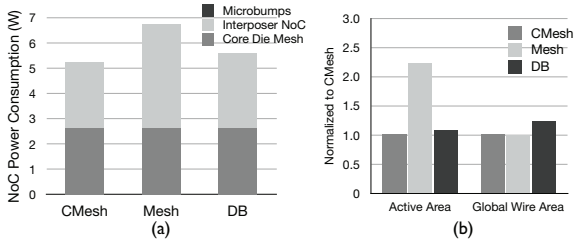

Fig. 4. (a) Power Breakdown for Core Die Mesh, Interposer NoC, and Microbumps. (b) Area Comparison for 1.75GHz operating frequency.

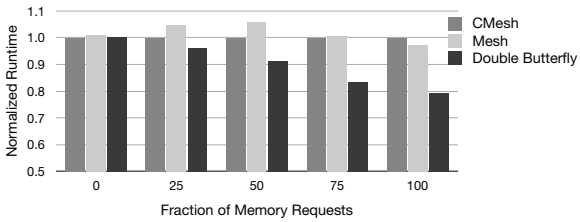

Fig. 5. Average completion time for uniformly distributed memory traffic.

significant due to the addition of extra metal layers. Compared to Mesh, active area is reduced as CMesh and DB require fewer total routers, which in turn also leads to overall power savings. Lower active area also makes these topologies more feasible for the passive interposer arrangement described in Fig. 3. The small black boxes in Fig. 2 at the middle of each group of four cores shows the die area needed to implement the logic for the interposer-layer routers (figure is drawn to scale).

VI. Performance Evaluation

This section provides a performance evaluation of our proposed NoC approach. We first compare DB against the Mesh and CMesh interposer topologies, followed by a variety of other indirect networks. Section VII then provides the simulation results for additional optimizations.

A. Uniform-Random Traffic

Fig. 5 shows the normalized average completion time of 1,000 messages by each core with a varying fraction of memory traffic. All results are normalized to the concentrated mesh (CMesh) on the interposer layer. The CMesh is chosen because it has the same router area/complexity and μbump overheads as the Double Butterfly (DB). For the results in this subsection, memory traffic is uniformly distributed across the 16 memory channels. For all evaluated workloads where the fraction of memory traffic is greater than zero, DB outperforms the other configurations. As the amount of memory traffic increases (i.e., high fraction of memory requests in the figure), the benefit of DB becomes more pronounced. The performance benefit comes from the reduced hop count in the routes from core-to-memory for DB.

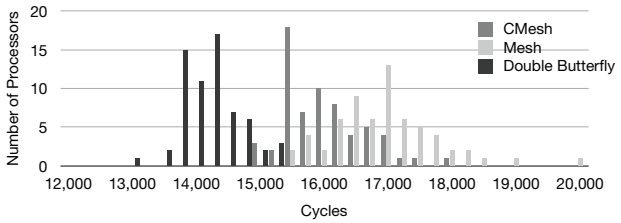

Fig. 6. Distribution of completion times where 25% of the total traffic consists of memory references, which are uniformly distributed.

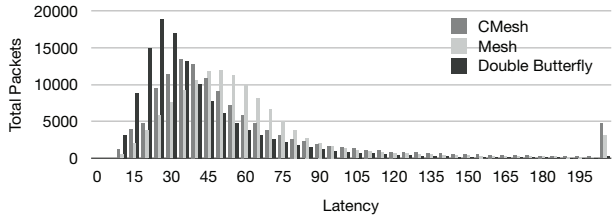

Fig. 7. Packet latency distribution with traffic consisting of 100% memory references, which are uniformly distributed.

The performance of CMesh initially improves compared to the unconcentrated Mesh, but as memory traffic increases, the performance improvement goes away (and CMesh is even slightly worse than Mesh at 100% memory load). The reason for this behavior is that at lower levels of memory traffic, the lower hop-count of CMesh provides a faster network transit time compared to Mesh. However, as network load increases, CMesh becomes bandwidth limited (recall that CMesh has only half of the bisection bandwidth of Mesh and DB), and then queuing delays rather than the number of hops dominate the end-to-end message latency. DB does well because it has a lower average hop count than either Mesh or CMesh while maintaining a bisection bandwidth equal to Mesh.

In addition to having a lower average completion time, DB also provides better fairness [34]. Cores in the network receive a more equal share of bandwidth to memory for DB. No explicit fairness mechanisms are employed; the balance is inherent to the DB topology due to the low hop count. Fig. 6 shows the distribution of completion times for all nodes with 25% memory traffic. CMesh and DB have tighter groupings of completion times indicating that all nodes receive a more equal share of bandwidth to and from the memory channels. The last cores to complete execution on the DB topology are as fast as the fastest cores on the unconcentrated Mesh. Similarly, we see the packet latency distribution is shifted to the left for DB versus Mesh and CMesh in Fig. 7. CMesh and Mesh have longer tails in these distributions; more packets in these networks experience severe congestion than in DB.

Fig. 8 shows the load-latency curves. In this experiment, we want to isolate the performance of the interposer network, so we assume 0% traffic on the CPU die. At low injection rates, the average latency for CMesh is better than the unconcentrated Mesh due to shorter average number of hops

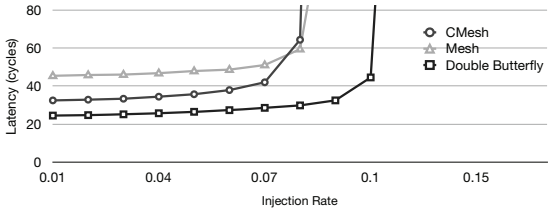

Fig. 8. Load-latency curves with uniformly-distributed memory references.

to reach the memory nodes. DB further improves average latency because the longer "diagonal" links further reduce the average path length. As discussed earlier, the reduced bisection bandwidth of CMesh causes it to saturate even before the unconcentrated Mesh. Mesh does not fare much better, despite having the same bisection bandwidth as DB. The reason is that the y-dimension links in the Mesh become a bottleneck, whereas DB provides greater path diversity so that requests that go to different "rows" in the network do not always use the same paths.

B. Non-Uniform Workloads

In addition to the basic uniformly-distributed memory traffic workload, Fig. 9 and Fig. 10 present results with the other less balanced/asymmetric traffic workloads, where 25% and 100% of the traffic goes to memory, respectively. For all of these experiments, DB significantly outperforms the other options with the exception of *Bisection* in the 100% memory traffic case. DB sees a modest performance degradation for *Bisection*; the static routing policy is not able to fully distribute the load in this pattern.[3] *UpperLeft* and *Corners* produce hotspots in the network that can lead to greater unfairness. We see the worst absolute performance for the *UpperLeft* traffic pattern across all topologies; DB is able to better distribute the hotspot loads and does not suffer as much as the other topologies. CMesh suffers from significant congestion and even performs worse than Mesh for *Corners*. DB provides more robust performance across a range of asymmetric traffic patterns leading to improved performance across a range of potential application/memory behavior.

Fig. 11 shows the distribution of completion times for the *UpperLeft* workloads which has the most imbalance. Compared to Fig. 6, we see an even wider spread of completion times. The standard deviation of completion times are 3060, 2337, and 782 cycles for Mesh, CMesh, and DB, respectively. As with the uniform workload analysis, DB clearly provides greater fairness than the other two topologies.

C. SynFull Evaluation

Fig. 12 compares the average packet latencies of the CMesh, Mesh, and DB networks using the SynFull traffic models [3].

[3]There is sufficient path diversity across the bisection for DB to perform well if adaptive routing is employed. The exploration of adaptive routing schemes is left for future work.

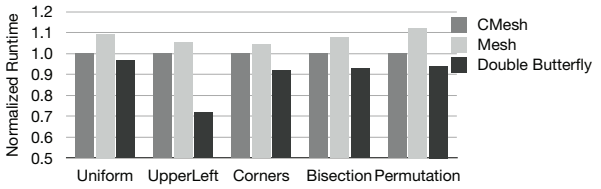

Fig. 9. Average completion time where 25% of total traffic consists of memory references.

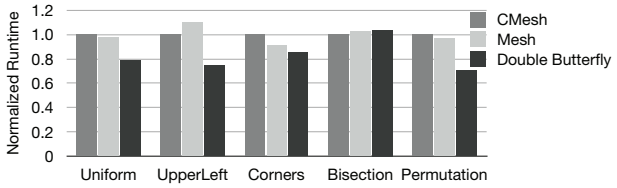

Fig. 10. Average completion time where 100% of total traffic consists of memory references.

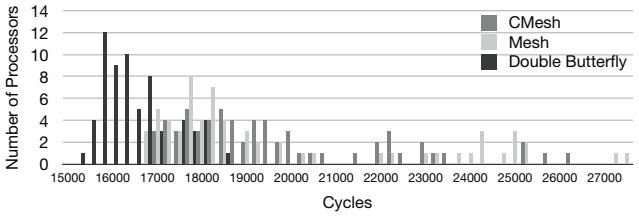

Fig. 11. Distribution of completion times with *UpperLeft* workload.

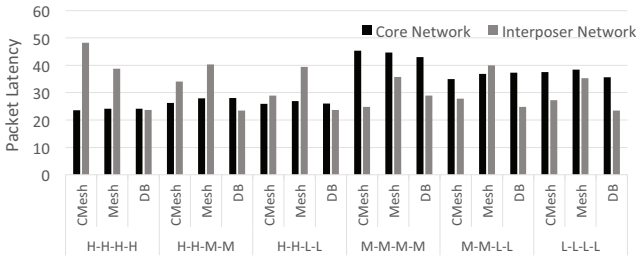

Fig. 12. Latency comparison using multi-programmed SynFull workloads.

SynFull workloads are combined based on memory intensity (e.g., the H-H-M-M category represents a multi-programmed workload constructed of two applications with high memory traffic and two applications with medium memory traffic). Multiple workload combinations are averaged in each category. We see consistently lower latency for DB across a range of memory intensities. The low hop count of the DB network gives it an advantage over the Mesh network. The performance of CMesh and DB are similar with DB having a slight advantage. CMesh performs the worst on the highest memory intensive workload, due to lower bisection bandwidth compared to the mesh and the formation of hotspots that cause significant congestion. These results confirm the conclusions reached in our extensive synthetic traffic evaluation.

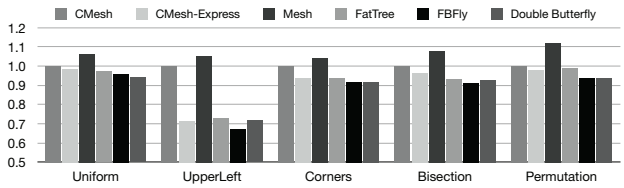

Fig. 13. Normalized runtime for additional topologies where 25% of the total traffic consists of memory references, which are uniformly distributed.

D. Other Topologies

Fig. 13 compares the DB network with other indirect topologies including a CMesh with express links, a fat tree, and a flattened butterfly. The fat tree and flattened butterfly (FBFly) achieve similar performance to the DB topology; however, the fat tree requires 4× more routers than in a conventional layout because it is used as an indirect network. The FBFly achieves competitive performance at the cost of larger and more power-hungry routers; FBFly routers have a degree of 12 versus 8 for the DB routers. Our proposed DB network is not the only topology that may make sense for the interposer, but the key take-away is to tailor the topology to the interposer traffic (i.e., middle-to-edge memory requests and not any-to-any coherence messages).

E. Summary

The Double-Butterfly topology is well suited to a multi-core system implemented on a silicon interposer. In a single-chip multi-core scenario, the single NoC layer must simultaneously support both core-to-core and main memory traffic, and so a relatively uniform topology such as a mesh performs reasonably well without introducing too much complexity.[4] However, by taking advantage of the additional routing resources on the interposer, the system designer can choose an interposer-layer topology specifically tailored for the core-to-memory requests.

VII. Interposer NoC Optimizations

A. Impact of Load Balancing

In Fig. 14, we consider the impact of using the interposer network for load balancing. If congestion is higher in the CPU die, coherence traffic can be moved to the interposer die. As mentioned in Section III-E, some restrictions are placed on which source-destination pairs are allowed to use the DB to prevent routing-level deadlock. No such restrictions are placed on the Mesh and CMesh networks as they are dimension-order routed. Mesh sees no benefit from load balancing across the various workloads for two reasons: the interposer-mesh does not reduce hop count, and Mesh has the worst fairness properties of the three designs. Load balancing does lower average packet latency but does not improve

[4]Our approach by no means limits the CPU layer to a mesh; other promising topologies such as a flattened butterfly could be considered.

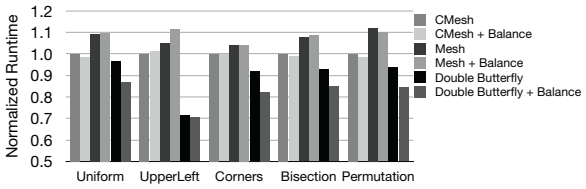

Fig. 14. Load balancing across layers with 25% of total traffic consisting of memory references.

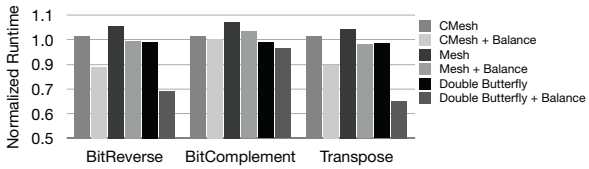

Fig. 15. Load balancing across layers with 25% of total traffic consisting of memory references and non-uniformly distributed CPU traffic.

total runtime as some cores suffer from greater contention due to longer hop counts to reach memory. CMesh sees very limited benefit from load balancing. CMesh provides reduced hop count, but again unfairness limits the overall runtime reduction that can be achieved. DB sees improvement across all workloads. The smallest improvement is seen in *UpperLeft*; this is expected because this pattern generates a significant hotspot that will dominate overall runtime.

The previous results used uniform-random traffic for the core die so that we can focus on the performance of the interposer die. However, with opportunities to off-load traffic from the CPU layer to the interposer, non-uniform core traffic should be considered. To that end, we simulate three popular synthetic traffic patterns in the multi-core die: bit reverse, bit complement, and transpose. These results are shown with uniform 25% memory traffic in Fig. 15. For unbalanced workloads like transpose, there is significant benefit to be had from off-loading traffic to the interposer die. In these scenarios, CMesh sees benefit from load balancing for bit reverse and transpose. These patterns have high average hop counts so they benefit from the reduced diameters of both CMesh and DB.

B. Impact of Using the Interposer NoC as Express Links

Fig. 16 shows the impact of utilizing express paths in the interposer layer for coherence traffic. We consider memory requests ranging from 10% to 50% of total traffic. The decision to use an express link is made statically. Independent of network load, if the interposer offers a shorter route (fewer hops) and does not require "doubling back," then the packet is routed on the interposer. When coherence messages dominate the traffic (e.g., 10% memory requests), the interposer's NoC slice is underutilized, and using the Double Butterfly's links as express routes provides over 8% benefit across the workloads. Not surprisingly, as the memory traffic increases,

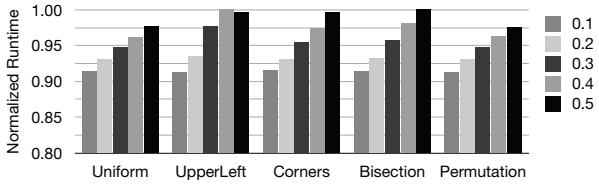

Fig. 16. Express routing with 10%-50% of total traffic consisting of memory references.

the benefit of the express links decreases as the coherence requests create network congestion for the memory traffic.

We also evaluated the express routing with non-uniform core traffic (plots are not shown due to space limitations). Like the load-balancing results, we observed dramatic improvement for transpose traffic. Some long paths in transpose effectively leverage the express links in DB for much lower hop counts. Bit complement's traffic pattern is such that it is not able to effectively leverage express paths.

VIII. Related Work

In this section we discuss related work in two areas: 3D NoC designs and multi-NoC designs. Brick and Mortar (BM) [33] proposes composable systems with different dies connected through an interposer layer. BM targets a greater degree of heterogeneity than our proposal. As a result, they design a highly configurable network to suit an unknown/unpredictable range of traffic demands [32]. The primary differentiation from our work and that of Kim *et al.* is that we do not require an active interposer, we target general traffic characteristics that persist across all applications (core-to-core and memory traffic), and we avoid the overheads associated with reconfiguration.

Recently, there have been numerous proposals for NoCs to leverage 3D die stacking [31], [36], [43], [53]. These networks are typically identical across all layers. No distinction is made between different types of traffic, but rather rich connectivity is sought across all layers of the stack. Die stacking allows the integration of different technologies; Morris *et al.* propose a symmetrical 3D optical NoC [41]. Xu *et al.* [51] customize each layer of a 3D NoC through long link insertion. This long-link insertion reduces hop count for all traffic. The Double-Butterfly network also uses long links but focuses on more efficiently routing memory traffic by using the under-utilized resources of the interposer rather than on reducing the diameter of a 3D network, which has greater wiring constraints due to the cost of additional metals layers. Kim *et al.* [26] consider the interconnect design space in the context of hybrid memory cube (HMC) based systems. Given a different set of system constraints, they decide to route all traffic through a memory-centric network leading to longer core-to-core latencies; alternatively, we functionally partition the network across the two layers and avoid penalizing core-to-core traffic.

Recent work proposes leveraging multiple networks to improve performance. For example, different types of coherence messages can be partitioned into multiple physical networks [48]. Multiple physical networks can obviate the need for virtual channels to break protocol-level deadlock. Recent work demonstrates that multiple physical networks are a more affordable solution that multiple virtual networks [52]. Tilera [49] has also taken this approach and separated traffic associated with coherence, memory traffic, I/O, etc. NOC-Out [40] targets specific Cloud workloads and assumes minimal core-to-core communication; their system is highly optimized for core-to-(L3-)cache (for I$ miss) performance. They use a butterfly to reach L3 slices and a different topology for the cores. Similarly, we use a Double Butterfly to optimize memory traffic, but the traffic patterns and layout of our system are different resulting in a new topology.

We follow a similar approach of functionally-partitioned NoCs, but there are several key differences. The first is the presence of the interposer. Messages will have to traverse the interposer layer to reach memory. Using the naive approach from Fig. 2(b) does not fully exploit the abundant resources in the interposer. Longer link topologies such as the DB or NOC-Out's butterflies may pose challenges to routing if implemented in a monolithic CPU die (as in the related works) necessitating additional, costly metal layers. Second, these NoCs do not support cross-layer load balancing due to protocol-level deadlock concerns. Furthermore, techniques that separate different cache coherence messages are complementary to our technique and can be added to optimize the CPU die, which is not the focus of this work. Balfour and Dally [5] explore some limited load-balancing across multiple networks. However, their networks are homogeneous in design. Catnap [14] explores the design of an energy proportional multi-NoC. These NoCs are not functionally partitioned, but rather turned on and off to respond to changes in network load.

Placement of memory controllers or channels within the NoC has been shown to have a significant impact on performance [1]. The Double-Butterfly design exploits similar observations: spreading traffic out across the multiple links used to reach the memory controllers results in superior performance. Both the Mesh and CMesh concentrate too much traffic on a small number of links while others remain idle; performance suffers as a result.

IX. Conclusions

Die-stacking technology (2.5D and/or 3D) provides fascinating new opportunities for re-imagining computer architectures. In this work, we explored one aspect of interposer-based multi-core systems, namely the interconnection network that ties the cores and memory together. We leverage otherwise unused routing resources for enhanced performance; given the presence of the interposer, these improvements come with little additional cost. By exploiting the different characteristics of coherence and memory traffic, along with the additional routing resources presented by the interposer, we have presented a general approach comprising a hybrid NoC architecture spanning both layers. The result is a NoC that achieves lower latencies, fairer response times, and higher sustainable bandwidth/throughput.

To the best of our knowledge, this work is the first to explore the design space of interposer-based NoC organizations. However, this is only a first step, as the overall problem space offers many more opportunities to research and explore. For example, the 2.5D stacking of multiple chips consisting of different combinations of CPUs, GPUs, other accelerators, DRAM, NVRAM, etc., can lead to many new NoC organizations, topologies, routing algorithms, and more. The possibilities for interposer-based systems are many, and we encourage others to explore what can be done with these die-stacking technologies.

Acknowledgment

The authors would like to thank the anonymous reviewers for their suggestions on how to improve this work. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Canadian Foundation for Innovation.

References

[1] D. Abts, N. Enright Jerger, J. Kim, D. Gibson, and M. Lipasti, "Achieving predictable performance through better memory controller placement in many-core CMPs," in *Intl. Symp. on Computer Architecture*, 2009.

[2] K. Athikulwongse, A. Chakraborty, J.-S. Yang, D. Z. Pan, and S. K. Lim, "Stress-Driven 3D-IC Placement with TSV Keep-Out Zone and Regularity Study," in *Intl. Conf. on Computer-Aided Design*, San Jose, CA, November 2010, pp. 669–674.

[3] M. Badr and N. Enright Jerger, "SynFull: Synthetic traffic models capturing a full range of cache coherence behaviour," in *Intl. Symp. on Computer Architecture*, June 2014.

[4] A. Bakhoda, J. Kim, and T. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," in *43rd Intl. Symp. on Microarchitecture*, Atlanta, GA, December 2010, pp. 421–432.

[5] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Intl. Conf. on Supercomputing*, 2006.

[6] V. E. Beneš, "Optimal Rearrangeable Multistage Connecting Networks," *Bell System Technical Journal*, vol. 43, pp. 1641–1656, 1964.

[7] C. Bienia, "Benchmarking Modern Processors," Ph.D. dissertation, Princeton University, 2011.

[8] N. Binkert *et al.*, "gem5: A Multiple-ISA Full System Simulator with Detailed Memory Model," *Computer Architecture News*, vol. 39, June 2011.

[9] B. Black, "Die Stacking is Happening," in *Intl. Symp. on Microarchitecture*, Davis, CA, December 2013.

[10] B. Black *et al.*, "Die-Stacking (3D) Microarchitecture," in *MICRO-39*, 2006.

[11] C. Clos, "A Study of Non-Blocking Switching Networks," *The Bell System Technical Journal*, vol. 38, no. 5, pp. 406–424, March 1953.

[12] J. Cong and Y. Zhang, "Thermal-Driven Multilevel Routing for 3-D ICs," in *10th Asia South Pacific Design Automation Conference*, Shanghai, China, January 2005.

[13] W. J. Dally, "Express cubes: Improving the performance of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016–1023, 1991.

[14] R. Das, S. Narayanasamy, S. K. Satpathy, and R. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," in *Intl. Symp. on Computer Architecture*, 2013.

[15] Y. Deng and W. Maly, "Interconnect Characteristics of 2.5-D System Integration Scheme," in *Intl. Symp. on Physical Design*, Sonoma County, CA, April 2001, pp. 171–175.

[16] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support," in *SC*, 2010.

[17] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express Cube Topologies for On-Chip Interconnects," in *Intl. Symp. on High Performance Computer Architecture*, Raleigh, NC, February 2009.

[18] J.-H. Huang, "NVidia GPU Technology Conference: Keynote," Tech. Rep., March 2013.

[19] R. Huemoeller, "Through Silicon Via (TSV) Product Technology," Amkor Technology, Tech. Rep., February 2012, Presented to IMAPS North Carolina Chapter.

[20] W.-L. Hung, G. Link, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Interconnect and Thermal-aware Floorplanning for 3D Microprocessors," in *7th Intl. Symp. on Quality Electronic Design*, San Jose, CA, March 2006.

[21] Institute of Microelectronics, "Process Design Kit (PDF) for 2.5D Through Silicon Interposer (TSI) Design Enablement & 2.5D TSI Cost Modeling," August 2012.

[22] M. Jackson, "A Silicon Interposer-based 2.5D-IC Design Flow, Going 3D by Evolution Rather than by Revolution," Synopsis Insight Newsletter, Tech. Rep., 2012, issue 1.

[23] JEDEC, "Wide I/O Single Data Rate (Wide I/O SDR)," http://www.jedec.org/standards-documents/docs/jesd229.

[24] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Intl. Symp. on Performance Analysis of Systems and Software*, 2013.

[25] T. H. Kgil *et al.*, "PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor," in *12th Symp. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 2006, pp. 117–128.

[26] G. Kim, J. Kim, J.-H. Ahn, and J. Kim, "Memory-centric system interconnect design with hybrid memory cubes," in *Intl. Conf. on Parallel Architectures and Compilation Techniques*, 2013.

[27] J.-S. Kim *et al.*, "A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking," in *ISSCC*, 2011.

[28] J. Kim, J. Balfour, and W. J. Dally, "Flattened Butterfly Topology for On-Chip Networks," in *Intl. Symp. on Microarchitecture*, Chicago, IL, December 2007.

[29] J. Kim, W. J. Dally, and D. Abts, "Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks," in *Intl. Symp. on Computer Architecture*, San Diego, CA, June 2007.

[30] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *Intl. Symp. on Computer Architecture*, Boston, MA, June 2006.

[31] J. Kim *et al.*, "A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures," in *34th Intl. Symp. on Computer Architecture*, San Diego, CA, June 2007.

[32] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin, "Polymorphic on-chip networks," in *Intl. Symp. on Computer Architecture*, 2008.

[33] M. M. Kim, M. Mehrara, M. Oskin, and T. Austin, "Architectural implications of brick and mortar silicon manufacturing," in *Intl. Symp. on Computer Architecture*, 2007.

[34] M. M. Lee, J. Kim, D. Abts, M. Marty, and J. W. Lee, "Probabilistic Distance-Based Arbitration: Providing Equality of Service for Many-Core CMPs," in *43rd Intl. Symp. on Microarchitecture*, Atlanta, GA, December 2010, pp. 509–519.

[35] C. Leiserson, "Fat-trees: Universal networks for hardware efficient supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 892–901, 1985.

[36] F. Li *et al.*, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," in *33rd Intl. Symp. on Computer Architecture*, Boston, MA, June 2006, pp. 130–141.

[37] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the Processor-Memory Performance Gap with 3D IC Technology," *IEEE Design and Test of Computers*, vol. 22, no. 6, pp. 556–564, November–December 2005.

[38] G. H. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in *ISCA-35*, 2008.

[39] G. L. Loi, B. Agarwal, N. Srivastava, S.-C. Lin, and T. Sherwood, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *43rd Design Automation Conference*, San Francisco, CA, July 2006, pp. 991–996.

[40] P. Lotfi-Kamran, B. Grot, and B. Falsafi, "NOC-Out: Microarchitecting a Scale-Out Processor," in *Intl. Symp. on Microarchitecture*, Vancouver, BC, December 2012, pp. 177–187.

[41] R. Morris, A. K. Kodi, and A. Louri, "3D-NoC: Reconfigurable 3D Photonic On-Chip Interconnect for Multicoresj," in *Intl. Conf. on Computer Design*, Montreal, Canada, September 2012, pp. 413–418.

[42] C. Okoro *et al.*, "Analysis of the Induced Stresses in Silicon During Thermocompression Cu-Cu Bonding of Cu-Through-Vias in 3D-SIC Architecture," in *the Electronic Components and Technology Conference*, Reno, NV, May 2007, pp. 249–255.

[43] D. Park *et al.*, "MIRA: A Multi-layered On-chip Interconnect Router Architecture," in *Intl. Symp. on Computer Architecture*, Beijing, China, June 2008, pp. 251–261.

[44] K. Puttaswamy and G. H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," in *ACM Great Lakes Symp. on VLSI*, Philadelphia, PA, May 2006, pp. 19–24.

[45] K. Saban, "Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwidth, and Power Efficiency," Xilinx, White Paper, 2011, wP380 (v1.1).

[46] M. Santarini, "Stacked & Loaded: Xilinx SSI, 28-Gbps I/O Yield Amazing FPGAs," Xilinx Xcell Journal, Tech. Rep., First Quarter 2011.

[47] C. Sun *et al.*, "DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *NOCS*, May 2012.

[48] S. Volos *et al.*, "CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache Coherent Servers," in *6th NOCS*, Lyngby, Denmark, May 2012, pp. 67–74.

[49] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the Tile Processor," *Micro, IEEE*, vol. 27, no. 5, pp. 15 –31, Sept.-Oct. 2007.

[50] Y. Xie, G. H. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architecture," *ACM Journal of Emerging Technologies in Computer Systems*, vol. 2, no. 2, pp. 65–103, April 2006.

[51] Y. Xu *et al.*, "A Low-radix and Low-diameter 3D Interconnection Network Design," in *15th Intl. Symp. on High Performance Computer Architecture*, Raleigh, NC, February 2009, pp. 30–42.

[52] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, "Virtual channels vs. multiple physical networks: a comparative analysis," in *Design Automation Conference*, 2010.

[53] A. Zia, S. Kannan, G. Rose, and H. J. Chao, "Highly-scalable 3D Clos NoC for Many-core CMPs," in *NEWCAS Conference*, Montreal, Canada, June 2010, pp. 229–232.